Ong Cheng Geng - Project Portfolio

Project: Horo (Calendar/Scheduling App)

Overview

Project Description

This is a portfolio that documents my contributions towards a Calendar and Scheduling application called Horo. Horo helps the user maintain a to-do list and to schedule their activities, and is able to post reminders to their desktop.

Here is a screenshot of our application in the default Calendar view.

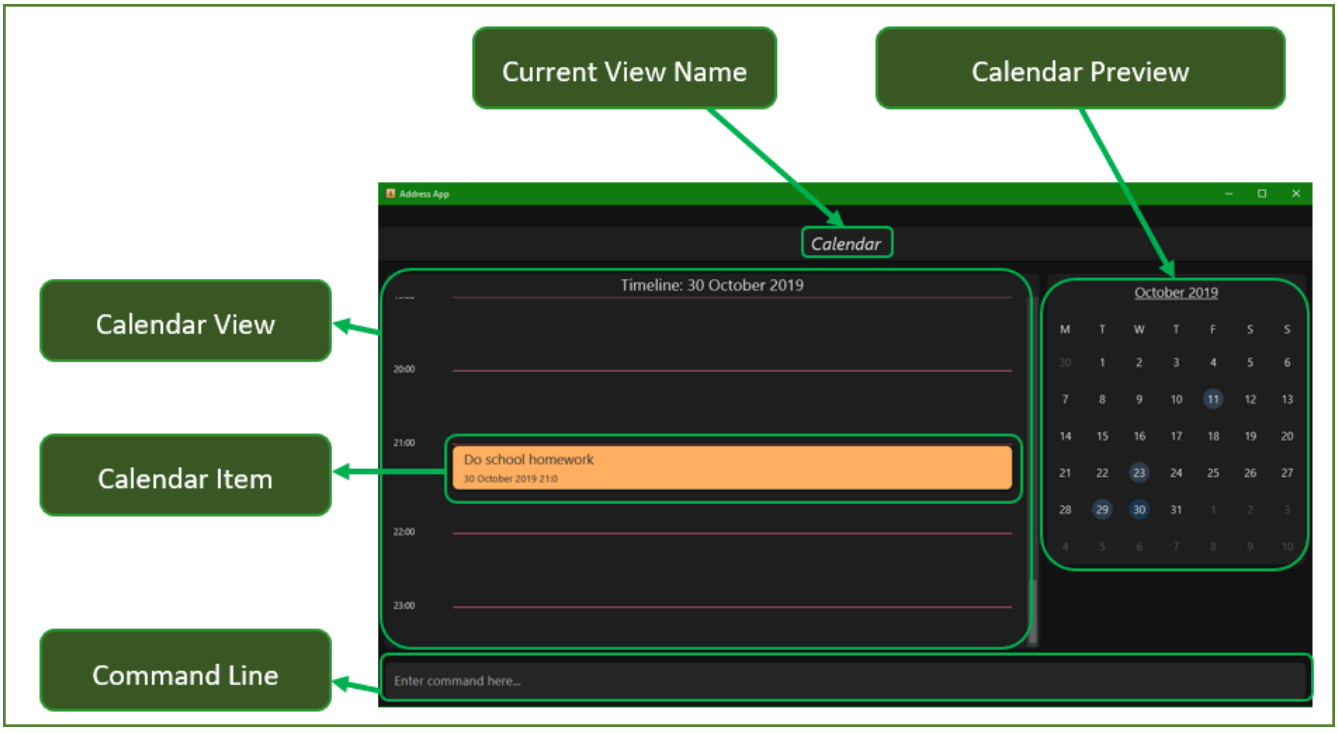

Figure 1. Horo Calendar view screenshot

Project Scope

Horo was made for the CS2103T Software Engineering module, offered by the National University of Singapore's School of Computing. Over the course of 6 weeks, students were grouped into teams tasked with either enhancing or morphing a basic Command Line Interface (CLI) application. I contributed to the project by writing both functional and test code, as well as documenting these contributions in the product's User and Developer guides.

Portfolio Purpose

My role in this project was to design and implement the Notification system of the application. Subsequent sections will go into greater detail with regards to my contributions to both the code and documentation.

Note the following symbols and formatting used throughout this document:

`notif_on`	A grey highlight in the User Guide indicates that a term is a command that can be executed by the program if input through the command line.
`NotificationManager`	A grey highlight in the Developer Guide indicates that a term is a class or an object used in the source code of the application.

Summary of Contributions

This section lists the code and documentation I contributed to the project.

Major Enhancement Added: Notification System

What it does	The Notification System runs in parallel to the main thread of the app. Every minute, it checks whether the user should be reminded of any Event or Task, and posts a new notification if necessary.
Justification	The Notification System is a natural extension of an application meant for tracking deadlines and schedules. It serves to remind a forgetful or busy user of their commitments.
Highlights	• An initial design for the notification system involved running it not as a parallel thread, but as a separate background application. This would allow notifications to be posted even if the main app were closed. However, this design was eventually reworked. Going forward with the original design would necessitate the development of a second application that the user would have to install in addition to Horo. This would have been at odds with module requirements and learning objectives. • An alternative design involved running instantiating the Notification System as a separate component to UI, Logic and Model. However, this design would have been at odds with OOP principles. The Notification System was thus further split into separate extensions of the Logic and UI packages.

Code Contributed

RepoSense Link

Notification System	1 2 3 4 5 6 7
Notification Tests	1 2 3
Task Tests	1 2
Task-Related Command Tests	1 2 1

Other Contributions

Project Documentation	Reviewed pull requests #78 #95 #107
Documentation	Edited the User Guide for better clarity and to fix grammatical mistakes #121

Contributions to the User Guide

The following section includes excerpts of my contribution to the Horo User Guide, explaining the Notification System and the `notif_on` and `notif_off` commands.

Contributions to the Developer Guide

The following section includes excerpts of my contributions to the Horo Developer Guide.

Class Architecture and Behaviour

This sub-section includes my explanations of the class architecture and behaviour behind the Notification System.

Notification System

Class Architecture

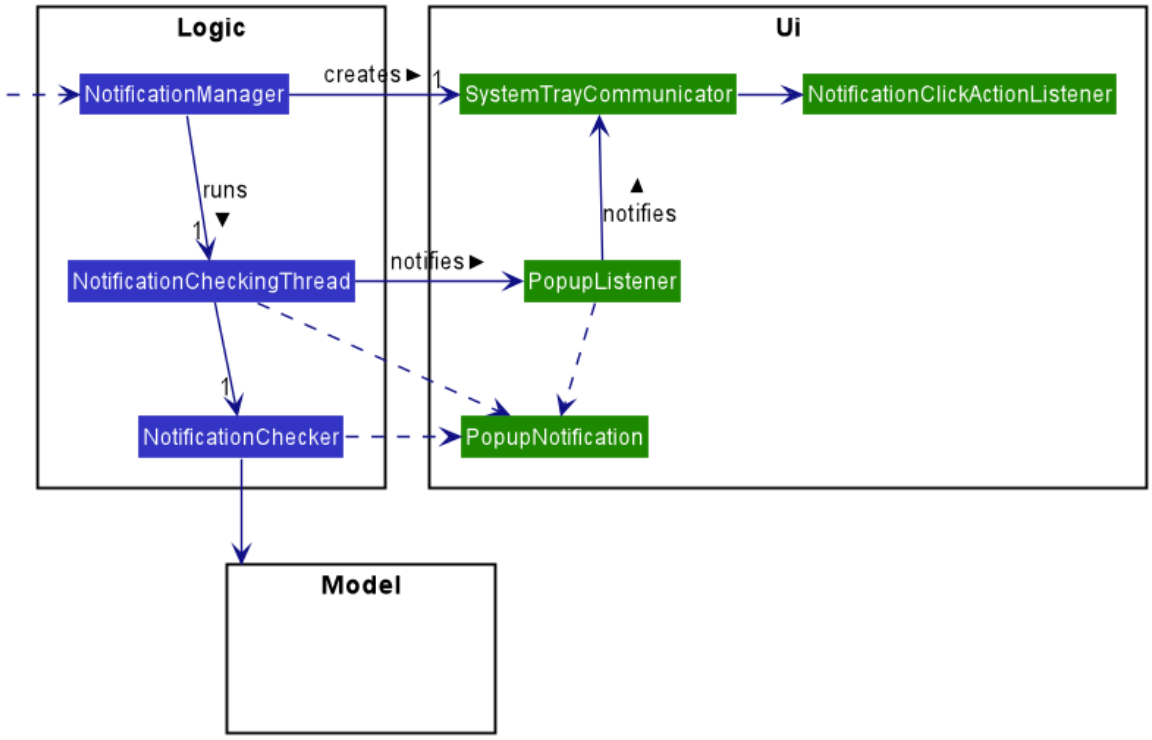

Figure 2. Class diagram for Notification System

The Notification System is facilitated by the `NotificationManager`, which is found in the Logic component. Other constituent classes of the Notification System can be found in the Logic and UI components, depending on their functionality. These classes and their functionalities are listed below:

Logic Classes

Logic classes are responsible for deciding if a notification should be posted. As with other components, their functionality is accessed through the `NotificationManager` class. The `NotificationManager` class maintains a reference to a `NotificationCheckingThread` as well as a `SystemTrayCommunicator`.

The logic classes of the Notification System can be found under the `notification` package under the `Logic` component.

- The `NotificationCheckingThread` is a daemon thread that runs in parallel with the main application. It checks for new notifications to post every minute.
- The `NotificationChecker` is responsible for checking `Model` for any notifications that need to be posted.

UI Classes

UI classes are responsible for displaying notifications to the user.

The UI classes of the Notification System can be found under the `systemtray` package under the `ui` component.

- The `PopupListener` class is the main channel of communication between the logic and UI classes.

When a notification needs to be posted, it will relay the information from the logic to UI classes.

- The `SystemTrayCommunicator` handles posting notifications and displaying the app's icon on the System Tray. It listens to the `NotificationCheckingThread` through a `PopupListener`.
- The `PopupNotification` class carries the information that will be posted to a popup notification.
- The `NotificationClickActionListener` is called when the user clicks on a popup notification.

Class Behaviour

As with other Manager classes, an instance of the `NotificationManager` is created upon the starting of MainApp. The NotificationManager proceeds to initialize and run a `NotificationCheckingThread`, as well as a `SystemTrayCommunicator`. Upon being started, the `NotificationCheckingThread` will enter a `notificationCheckingLoop` by calling its method of the same name.

To give a better explanation of how the `NotificationCheckingThread` works, a single run of its loop is illustrated below:

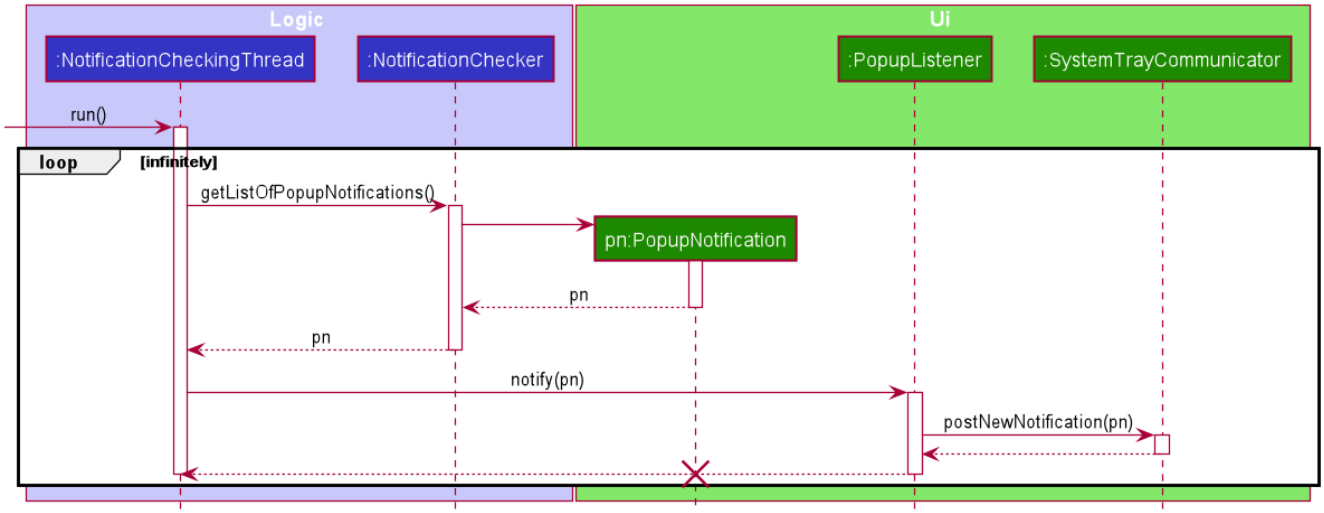

Figure 3. Sequence diagram for NotificationCheckingThread's main loop

Step 1. The `NotificationCheckingThread` calls the `NotificationChecker` to generate instances of `PopupNotification` through a call to `NotificationChecker#getListOfPopupNotifications()`

Step 2. For each `PopupNotification` generated by the NotificationChecker, a call to `PopupListener#notify()` is made.

Step 3. This prompts the `SystemTrayCommunicator` to post a new notification.

Step 4. The `NotificationCheckingThread` sleeps until the start of the next minute, found by the method `NotificationCheckingThread#findMillisecondsToNextMinute()`.

Design Considerations

Aspect: How the Notification system should run

- **Alternative 1 (current choice):** Running the Notification system as a separate thread in the same application

    - Pros: Easier to implement and test.
    - Cons: The user would have to leave the application on if they always wanted to be notified.
- **Alternative 2:** Running the Notification system as a background application
    - Pros: This would allow notifications to be posted to the user's desktop even if the Horo main app were not open.
    - Cons: This would require the creation of a separate application that the user would have to install on their computer. Because different Java applications are ran in different instances of Java Virtual Machines, this could vastly complicate implementation as the Notification System and the rest Horo would be unable to interact directly.

Alternative 1 was eventually chosen as it was simpler to implement and test, and remain within the initial scope of Horo's development. The application can be potentially changed to use Alternative 2 in the future.

Intructions for Manual Testing

This sub-section includes instructions I have provided for manual testing of the notification system.

Notification System

1. Posting notifications to the desktop
    a. Prerequisites: Make sure notifications have been switched on by using the `notif_on` command.
    Make sure the System Tray is supported.
    b. Test case: `add_event "Test Event" "[CURRENT DATE] [CURRENT TIME INCREMENTED BY ONE MINUTE]"`
    Expected: Upon the next minute, a notification should be posted to your desktop through the system tray.
    c. Test case: `add_task "Test Task" --due "[CURRENT DATE] [CURRENT TIME INCREMENTED BY ONE MINUTE]"` Expected: Upon the next minute, a notification should be posted to your desktop through the system tray.